ModelLib: A Web-Based Platform for Collecting Behavioural Models and Supporting the Design of AMS Systems

Torsten Mähne*, Alain Vachoux†
Laboratoire de Systèmes Microélectroniques (LSM)
Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Abstract

This paper describes ModelLib, a web-based platform for collecting models from different domains (e.g. electrical, mechanical) and levels of abstractions. Use cases for this tool are presented, which show how it can support the design process of complex AMS systems through better reuse of existing models for tasks like architecture exploration, system validation, and creation of more and more elaborated models of the system. The current state of the implemented ModelLib prototype is described and an outlook on its further development is given.

Keywords: AMS design process; model library; model meta information; relational database; Apache; PHP; Subversion; wiki.

1 Introduction

The fast progressing advances in manufacturing technology allow the integration of more and more functionality from different disciplines into a single complex system. This leads to a continuously growing in the needed design effort where at the same time product cycles get shorter. The resulting increase in the "design productivity gap" is especially notable in semiconductor industry. There the technological production capacity (measured by the number of available transistors) has increased since 1985 yearly between 41 % and 59 % whereas the the design capacity (measured by the efficient use of transistors) has increased only at a yearly rate of 20 % to 25 % [OTD+05]. To allow the control of the design costs and prevent them to get prohibitive expensive, new design technologies have to be continuously introduced, like in the past block reuse or IC implementation tools.

The design of heterogeneous Analogue and Mixed-Signal (AMS) systems is still a highly manual work, which is currently only rudimentarily supported by EDA tools and not as automatised as the design of digital systems using logic synthesisers and place & route tools. An efficient tool support for the AMS design flow is missing. Specialised simulators for different physical disciplines and levels of abstraction as well as CAD tools to design and layout the physical realisation are used for each component of a heterogeneous system. These tools originate from different engineering fields, which leads to problems when exchanging models and other design data between them. For example, the design of a typical Micro-Electro-Mechanical System (MEMS) like an inertial yaw rate sensor requires, among others:

- optimisation and characterisation of the micromechanical resonator and (separately) of the electrostatic field distribution of the comb drive structures, which are driving and sensing the movement of the flexible structure, using an FE analysis tool like ANSYS™ from mechanical engineering;
- simulation of the whole system on the circuit level taking into account the coupling between the mechanical and electrostatic domain within the MEMS transducer and feedback from the analogue and digital driving and sensing circuits using behavioural simulators and modelling languages like VHDL-AMS coming from electrical engineering;
- layout of the mechanical structure and electronic circuits using IC layout tools.

The challenge for the EDA industry in the short term is to improve the links between the existing tools. Here one research area, which relates to the given yaw rate sensor example, are reduced-order modelling methods that allow the extraction of fully coupled behavioural models for circuit level simulation from detailed FE models of the device [MKF+05]. In the long term new design methods and integrated tool chains are needed to support the designer in the whole process of specification, design, integration, validation/verification, and integration of the different components of a complex AMS system. First approaches for the specification, synthesis, and automated layout generation exist for moderate-complexity analogue circuits (device count less than 100), e.g., the *AMGIE* approach and the *Mondriaan* tool described in [VGS02].

One important aspect of an AMS design flow is the management of the models of a device, component, or

*torsten.maehne@epfl.ch
†alain.vachoux@epfl.ch

whole system on the different levels of abstraction. They are created during the different stages of the design process to simulate their future behaviour and verify that the specified requirements are met. To speed up the creation of these models it is advantageous to reuse existing models and to possibly adapt them further. Nowadays the designers usually reuse only their own models or those provided by the design environment. An exchange of the models between designers is complicated by the fact that they are usually not aware if and where a similar model is already existing. Furthermore these models need to be documented regarding their interface, implementation, extend of covered effects, how they were verified, and other general properties. The Engineering Society For Advancing Mobility Land Sea Air and Space International (SAE) covers these documentation issues with the *Model Specification Process Standard* [SAE02]. There are also ongoing activities developing collections of verified models for different design languages like *Modelica* [Mod06] and *VHDL-AMS* [HPH+05]. Those libraries can often be downloaded as archives from the Internet and provide the model source code and some documentation, which can be to some degree automatically extracted from the model sources using a documentation generator.

There are also tool-specific library managers, such as the *Library Manager* in the Cadence™ IC Design Environment or the Workspace in Mentor Graphics™ Model*Sim*, to handle the various models of a project and the design-kits. However, these tools cannot cope with the management of the models over the tool boundaries as it is required for heterogeneous AMS system designs. Handwritten websites documenting and linking to model archives are one solution, but can only partly address these issues and require a lot of manual maintenance to stay up-to-date. The *ModelLib* project addresses these problems through the development of a web-based platform for collecting these models, making them available over the Internet, and supporting their collaborative further development.

This paper describes the ModelLib platform that is being developed to address the described needs, but which will also offer features supporting the design of complex heterogeneous AMS systems. Section 2 presents the basic use cases for a model library and how it can support the work of the AMS designer. From this the requirements for the ModelLib platform are developed. The architecture of a prototype implementation is described in Section 3. The conclusions are given together with an outlook on further developments in Section 4.

2 Use Cases and Requirements

A model library like ModelLib can be set up on different organisational levels, like within an project group, a company, or as a community portal on the Internet. The basic use cases for the AMS designer accessing the ModelLib server through a client on his computer for submitting, retrieving, and collaboratively developing the models over the Internet remain the same, while the demands for security and required detail of access control will rise with each level of broader access. First the use cases for retrieving a model (through browsing or a more complex query) are presented, since they show which meta information need to be stored in the library alongside the models to support the designer's decisions, and then the remaining use cases regarding submission, updating, and jointly developing models.

One way to access the models in the library is to directly browse through the collection of available models. For this it is required to sort them into a hierarchy of model classes, where a model can be at the same time member of different classes, e.g., to reflect that a model is part of some Intellectual Property (IP) library and modelling effects of a particular physical domain. After selecting a model the user is presented the meta information describing the properties of the model, which can be roughly detailed into two groups answering the following questions: "How the model is built?" and "How the model can be used?". The first group describes the following aspects:

- name and storage place of the model within the model class hierarchy;
- interface, including detailed information about all parameters and ports as well as the made assertions;
- one or more model implementations (architectures) in the form of behavioural descriptions and/or structural descriptions along with the made assertions;
- design entities, each one gathering the interface and one model implementation using a particular design language (e.g. VHDL-AMS) and tested against particular tools (e.g. simulators);
- testbenches, which are stored alongside the models and refer to some design entities. They are also implemented using some design language version and stored in a number of files, which are known to work together with some design tool versions. Test data and expected results can be also included.

To each of these aspects an arbitrary number of references to external documents can be given. These are information, which can be directly extracted from the model sources. To support the porting of design entities and testbenches to other design tools, it is required to store also information about the different versions of available design languages and design tools as well as which tools support which language.

The second group of meta information further characterises a model regarding its fidelity, performance, and use by describing the following aspects, as detailed for example in the SAE J2546 Standard [SAE02]:

- level of model refinement: *Pins* (named interface but no internal features), *Static* (time-invariant, steady state internal behaviour), *Dynamic* (time-varying behaviour), *Precision* (including significant amount of second order effects), *Vector* (model with directional or spatial interface);
- to which extent the different features of a model are implemented (levels 0–7 in SAE J2546): *None* (not included), *Named* (acknowledged but unimplemented), *Fixed* (adjustment only possible through editing the model or an non-related parameter), *Index* (offers discrete choice of discrete values or modes), *Static* (accepts parameter value prior simulation run), *Dynamic* (adapting to internal conditions during simulation), *Mutual* (adapting to external influences during simulation), *Directional* (adapting to directional external influences during simulation);
- supported analysis types, like *Continuity* and *Loads Analysis*, *Nominal Analyses* (DC, transient, small signal, and stability analysis), *Stress Analysis*, *Perturbation* or *Sensitivity Analyses*, *Worst-Case Analyses*, *Failure Modes and Effects Analyses (FMEA)*, or *Sneak Circuit Analysis (SCA)*;
- validity of the model (under which assumptions and operating conditions it is valid);
- properties of the modelled component, e.g., gain, bandwidth of an amplifier, and, if applicable, how these properties can be characterised through testbenches;
- forms of simulation results regarding a specific property, like *Flag*, *Message*, *Scalar*, *Waveform*, or *Relation* (non-time series describing the interaction of two or more variables).

It is possible to include all this supplementary information into free-form description fields, but for large model collections, like they are intended for ModelLib, it is better to structure them as far as possible and implement them in the database scheme. This has to be done as general as possible, because not all properties, which a designer might think of storing for a model, are known in advance. It allows the AMS designer to better manage the models for different levels of abstraction of the same physical component.

A large collection of models also requires a more efficient access to the models besides browsing to support the designer in selecting the model with the right fidelity for reuse in the design tasks (e.g. architecture exploration, detailed component characterisation, system validation). To do that, the designer has to send queries to the model library. In simple cases this means querying for a *model name*, *keywords*, and full text search in the description fields of interface, architecture, etc. If this is not sufficient enough, more complex queries for certain properties of the model to be in a specified range (e.g. detail level, modelled effects, design language, component properties) can be made.

The previous use cases for browsing and querying of models showed which information has to be provided by the model developer when submitting a new model to the library. First, the files containing the source code, testbenches, test data, simulation results, and other documents of the model must be made accessible through Uniform Resource Locators (URLs), preferably by submitting them to the repository under the control of a revision control system. This should ease their further (cooperative) development and keep all model revisions accessible in the future. Then the meta information about the model, testbenches, etc. needs to be extracted from these files and entered into the library using structured input forms. This process can be partially automatised using similar techniques like the ones used by documentation generation tools [HPH+05]. During the further development of the model, this information needs to be continuously updated to keep it in sync with the changes made. The library supports the development process by providing a platform for discussing the model and jointly improving its implementation and documentation.

3 Prototype Implementation

Several software components have to be integrated to meet the requirements arising from the use cases presented in Section 2. The core component of ModelLib is a database that stores the meta information about the models. All files that contain the models and their accompanying documents are managed by a revision control system to allow their collaborative development. A wiki provides an open platform to discuss the models and collaborate on the improvement of their implementation and documentation.

A running prototype of ModelLib [Mäh06] is being implemented, which is using *PostgreSQL* [Pos06] to manage the database that stores the meta-information, *Subversion/WebSVN* [CSFP04] to handle the model repository, *YaWiki* [Jon05] to discuss and jointly develop the documentation of the models, and *PHP* [PHP06] to implement the web interface served out by an Apache 2 web server [Apa05]. This section describes the current state of the prototype.

Figure 1 shows the architecture of the prototype and how its different components interact. The lower part of the figure shows the different user created documents, which are managed by the ModelLib server. The file revisions of the documents are stored in the *repository*. The meta information about the models and accompanying documents are stored in the *meta information database*. Informal texts, like discussions and HOWTOs, are stored in the *wiki database*. On the server side implemented user interfaces provide the access to these three storages. This has the advantage that the user can access ModelLib over the Internet using a standard

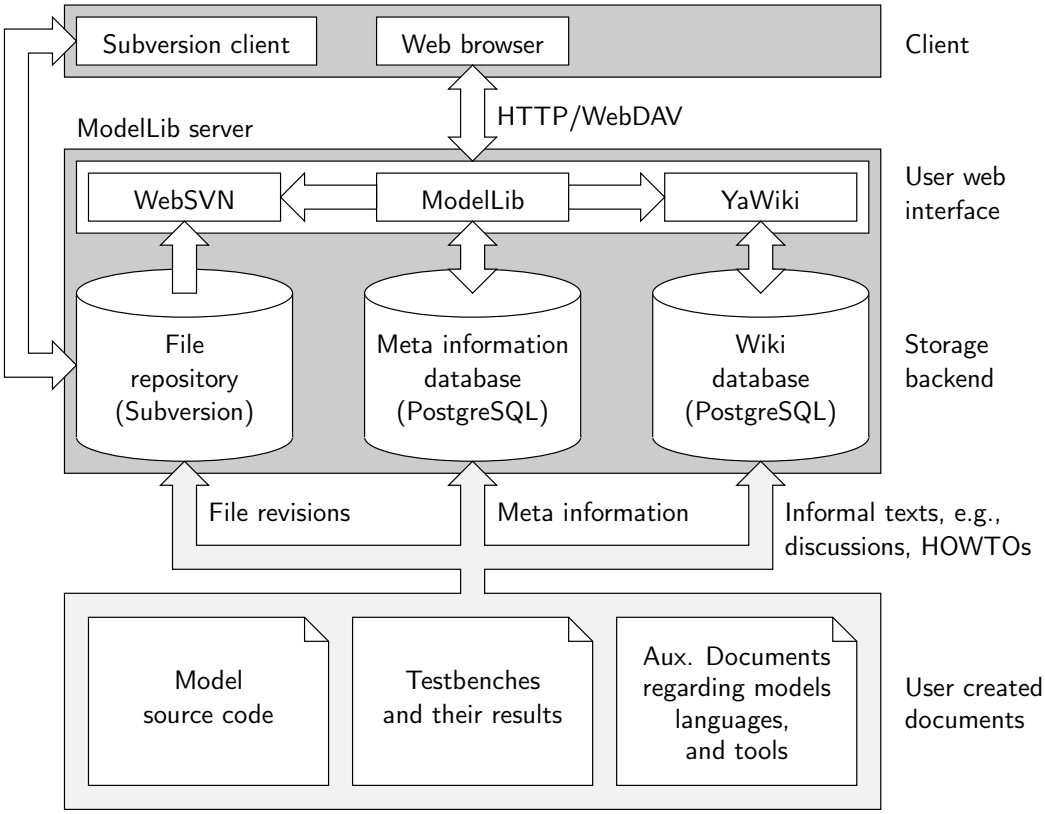

Figure 1: Architecture of the ModelLib prototype and the communication links between the components

web browser. The file revisions of the models and documents are managed on the user side by the Subversion client, which also provides the access to commit them to and update them from the repository.

The meta information describing the properties of the models is stored in a relational database. Figure 2 shows the Entity-Relationship (ER) diagram of the database that has been designed for the prototype. It currently considers the model class hierarchy, the information about referenced external documents, the information about available design language and tool versions, and the meta information about interface, architectures, assertions, design entities, testbenches, and files of the models. This fully covers the meta information from the first group "How the model is built?" described in Section 2. The meta information from the second group "How the model can be used?" is not yet implemented in the database structure and can currently only be stored as free-form descriptions.

The ModelLib web interface provides access to the model collection over the Internet. It queries the meta information about the models, testbenches, design languages, design tools and document references in the Relational Database Management System (RDBMS) and outputs it to a CSS formatted HTML page. In the prototype it is implemented in the server side scripting language PHP using specialised library packages from the public PEAR repository [PHP05], which ease the development of web applications. The ModelLib prototype makes for example use of `DB` and `Text_Wiki` to access the database through the PostgreSQL server and format the HTML output of the free-form description fields using a wiki-like syntax. Currently the web interface implements the following features: logging in as different database users; browsing the model class hierarchy for a model; displaying and editing of all information related to the interface, architectures, design entities, and testbenches of a model (Figure 3); displaying of the information about the available design languages and their different versions (Figure 4) as well as the one about the available design tools and their different versions (Figure 5); and displaying/editing of the document references.

The files containing the models, testbenches and their accompanying documents are stored in a repository that is managed by a revision control system to allow their collaborative development. The meta information database refers to the files in the repository using URLs. The *Subversion* system has been chosen for the ModelLib prototype since it provides most features of the widely used CVS and overcomes some of its known drawbacks by supporting, among others, versioned directories, renames and meta-data; truly atomic commits; efficient handling of binary files; and more efficient handling of tags and branches. An Apache 2 web server using the custom module `mod_dav_svn.so` from Subversion makes the repository available to the clients via the WebDAV/DeltaV protocol, which is an extension to

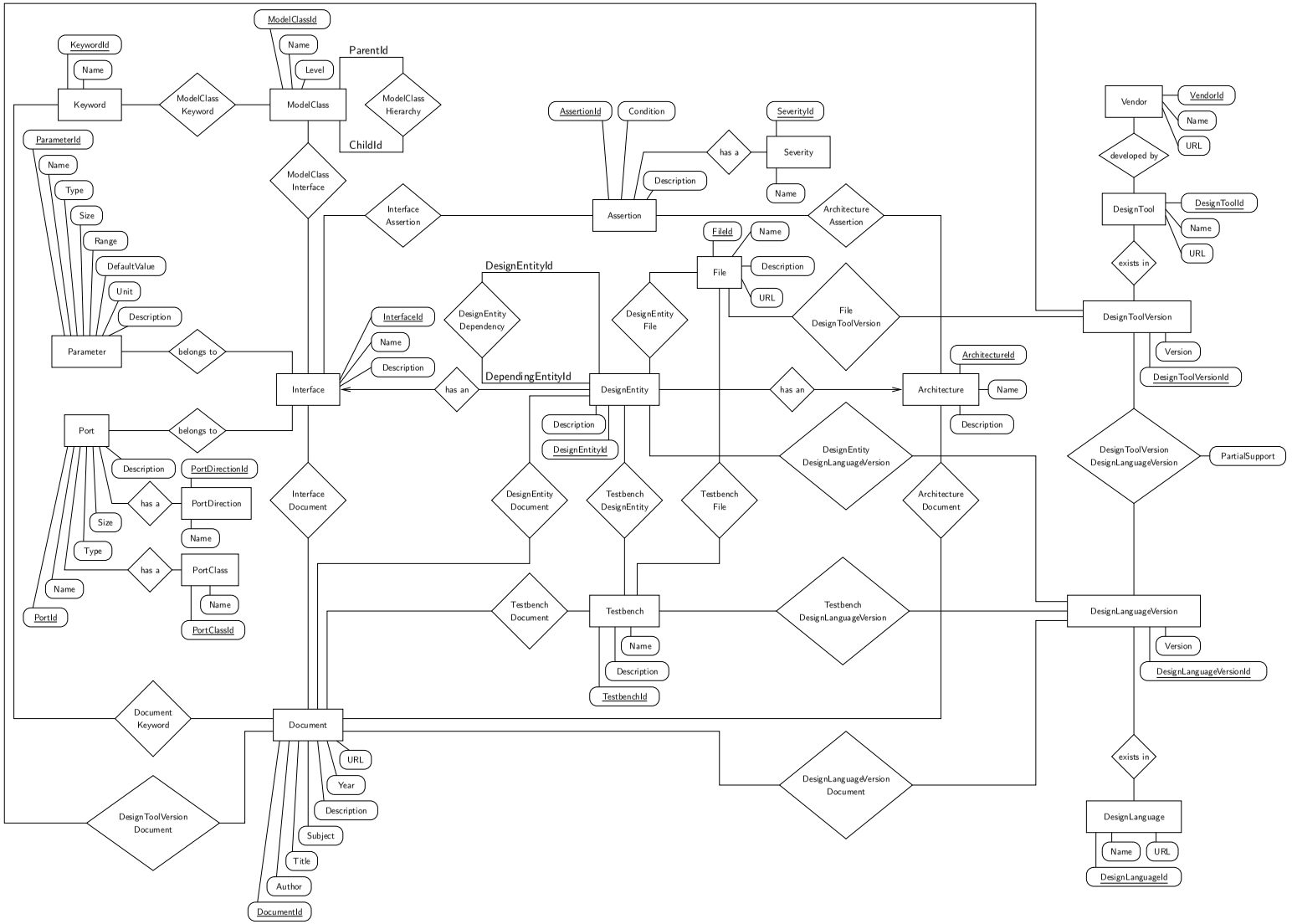

Figure 2: Entity-Relationship diagram of the meta information database

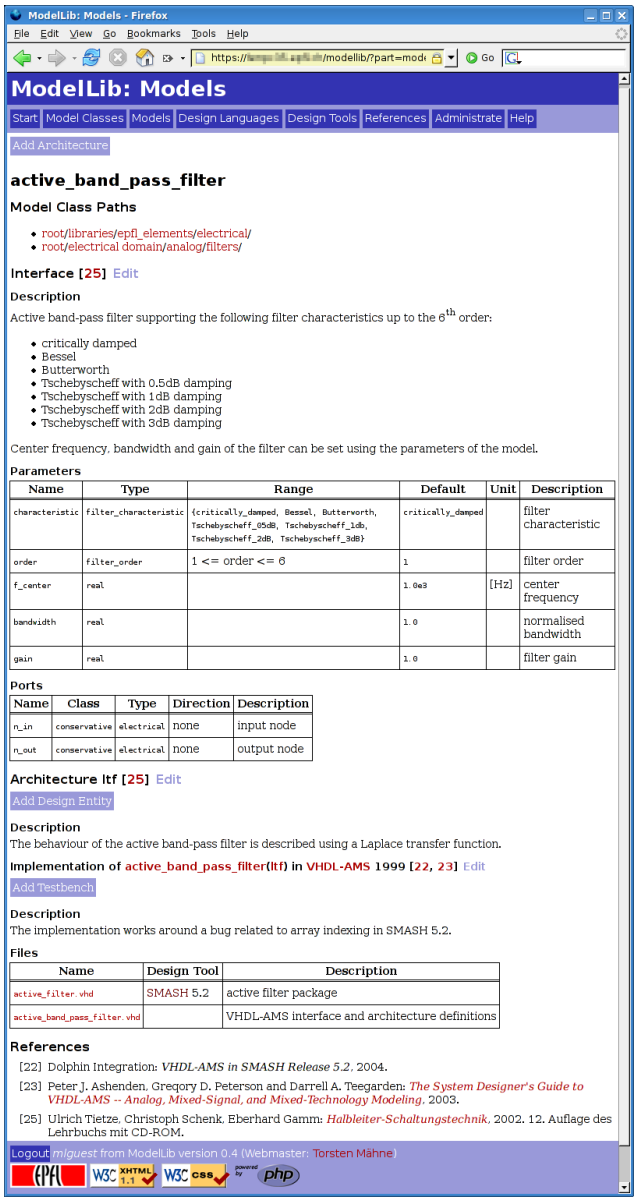

Figure 3: Meta information about a selected model

the well-known HTTP 1.1 protocol that adds versioned writing capabilities. This provides key features like authentication, path-based authorisation, wire compression, and basic repository browsing using a web browser or WebDAV client. The PHP based web interface *WebSVN* improves the browsing of the repository via a web browser considerably with features like viewing the file/directory logs; listing of all changed, added, or deleted files in queried revision; log message searching; Blame support; Tar ball downloads; Directory comparisons; and RSS Feed support.

A wiki provides an open platform to discuss the models and to collaborate on their implementation and documentation. An access right management has to be established to control the read and write access to the different models. The *YaWiki* engine has been chosen for the ModelLib prototype because it adds logical

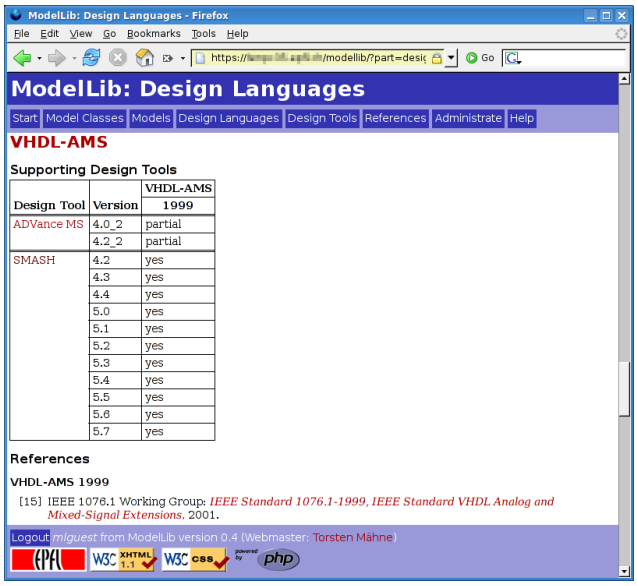

Figure 4: Meta information about design languages

name spaces, Access Control Lists (ACLs), navigational elements, and more to the traditional wiki; each side can be instantly commented through a web form on the wiki page; it is written in PHP like the other parts of the ModelLib web interface; its formatting engine is a PEAR module; which is also used in the ModelLib web interface to format the free-form description fields.

4 Conclusions and Outlook

The current ModelLib prototype presented in Section 3 implements basic features of a model library as described in Section 2. Its web interface allows browsing for a model through the model class hierarchy. The meta information about models, testbenches, design languages, design tools, and external documents that are stored in the meta information database can be displayed. New models can be added by committing them into the repository and adding/editing their meta information in the meta information database using the web interface.

The main task for the further development of the ModelLib prototype is to support the structured storage of the second group of meta information that characterise the fidelity, usage, performance, and other properties of the model. This implies an extension of the current database scheme. The querying of models is currently only supported through direct SQL queries to the meta information database. This use case needs to be implemented into the web interface, so that it will offer the designer elaborated query schemes to guide him/her in the selection of a suitable model for his current task. It is also important to address the usability and security of ModelLib by improving the implementation of the web interface to complete its functionality and to offer to the user a coherent interface to the

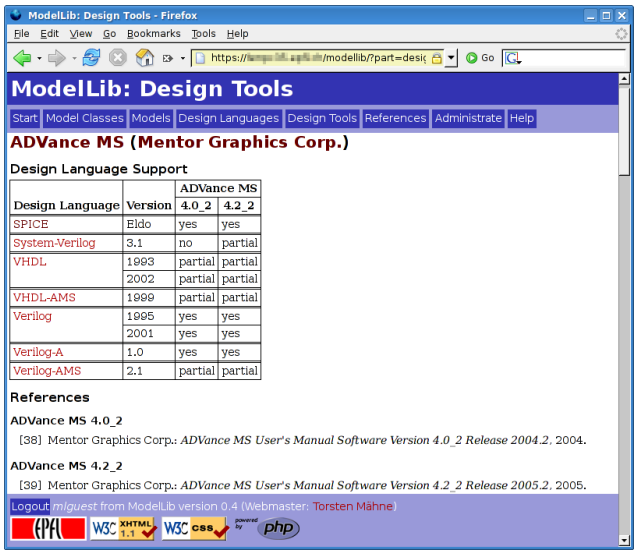

Figure 5: Meta information about design tools

three main components repository, meta information database, and wiki. This includes the implementation of a unified authentication and a fine-grained access control mechanism using ACLs to allow the usage of ModelLib by a broad audience over the public Internet and, at the same time, keeping control on who has read and write access to which part of the library. The efficiency of the model import and update can be improved by automatising the input of the meta information in the ModelLib database by extracting the information from the model source code.

In the near future the approach presented in this paper shall be validated through the support of practical design cases such as the design of RF transceivers and multi-channel A/D converters.

Bibliography

[Apa05] The Apache Software Foundation: *The Apache HTTP Server Project*, 1999–2005. http://httpd.apache.org/, visited on 27/03/2006.

[CSFP04] Collins-Sussman, Ben, Brian W. Fitzpatrick, and C. Michael Pilato: *Version Control with Subversion*. O'Reilly Media, Inc., Sebastopol, USA, 1st edition, June 2004. http://svnbook.red-bean.com/, visited on 27/03/2006.

[HPH+05] Hessel, Ewald, Klaus Panreck, Joachim Haase, André Schneider, and Steffen Scholz: *Development of VHDL-AMS-libraries for automotive applications*. In *Forum on Specification and Design Languages (FDL) 2005*, pages 101–110, September 2005. http://www.ecsi.org/fdl/, visited on 10/03/2006.

[Jon05] Jones, Paul M.: *YaWiki: Yet Another Wiki for PHP*, 2005. http://www.yawiki.com/, visited on 27/03/2006.

[MKF+05] Mähne, Torsten, Kersten Kehr, Axel Franke, Jörg Hauer, and Bertram Schmidt: *Creating virtual prototypes of complex micro-electro-mechanical transducers using reduced-order modelling methods and VHDL-AMS*. In *Forum on Specification and Design Languages (FDL) 2005*, pages 209–221, September 2005. http://www.ecsi.org/fdl/, visited on 10/03/2006.

[Mod06] The Modelica Association: *Modelica: Modeling of Complex Physical Systems*, 1997–2006. http://www.modelica.org/, visited on 30/03/2006.

[Mäh06] Mähne, Torsten: *Modellib prototype*, July 2006. https://lsmpc4.epfl.ch/modellib/, visited on 13/07/2006.

[OTD+05] O'Connor, I., F. Tissafi-Drissi, *et al.*: *UML/XML-based approach to hierarchical AMS synthesis*. In *Forum on Specification and Design Languages (FDL) 2005*, pages 89–100, September 2005. http://www.ecsi.org/fdl/, visited on 10/03/2006.

[PHP05] The PHP Group: *PEAR: The PHP Extension and Application Repository*, 2001–2005. http://pear.php.net/, visited on 27/03/2006.

[PHP06] The PHP Group: *PHP: Hypertext Preprocessor*, 2001-2006. http://www.php.net/, visited on 27/03/2006.

[Pos06] PostgreSQL Global Development Group: *PostgreSQL: The World's Most Advanced Open Source Database*, 1996–2006. http://www.postgresql.org/, visited on 27/03/2006.

[SAE02] SAE Electronic Design Automation Standards Committee: *SAE J2546: Model specification process standard*. Technical report, SAE: The Engineering Society For Advancing Mobility Land Sea Air and Space International, 400 Commonwealth Drive, Warrendale, PA 15096-0001, USA, 2002.

[VGS02] Van der Plas, Geert, Georges Gielen, and Willy Sansen: *A Computer-Aided Design and Synthesis Environment for Analog Integrated Circuits*. Kluwer Academic Publishers, Boston, Dordrecht, London, 2002.